Tarea 3

Análisis y Diseño de Algoritmos

Rafael Hermida Toledo

ID: 8964994

Marzo 12, 2024

Problemas conceptuales

1) Ejercicio 2: *Text Segmentation* (Erickson, página 124).

   Describe efficient algorithms for the following variants of the text segmentation problem. Assume that you have a subroutine $IsWord$ that takes an array of characters as input and returns *True* if and only if that string is a "word". Analyze your algorithms by bounding the number of calls to $IsWord$.

   (a) Given an array $A[1...n]$ of characters, compute the number of partitions of A into words. For example, given the string *ARTISTOIL*, your algorithm should return 2, for the partitions $ARTIST \cdot OIL$ and $ART \cdot IS \cdot TOIL$.

   (b) Given two arrays $A[1...n]$ and $B[1...n]$ of characters, decide whether A and B can be partitioned into words at the same indices. For example, the strings *BOTHEARTHANDSATURNSPIN* and *PINSTARTRAPSANDRAGSLAP* can be partitioned into words at the same indices as follows:
   $BOT \cdot HEART \cdot HAND \cdot SAT \cdot URNS \cdot PIN$
   $PIN \cdot START \cdot RAPS \cdot AND \cdot RAGS \cdot LAP$

   (c) Given two arrays $A[1...n]$ and $B[1...n]$ of characters, compute the number of different ways that A and B can be partitioned into words at the same indices.

2) Ejercicio 6: *Suffle of Strings* (Erickson, página 126).

   A shuffle of two strings X and Y is formed by interspersing the characters into a new string, keeping the characters of X and Y in the same order. For example, the string *BANANAANANAS* is a shuffle of the strings *BANANA* and *ANANAS* in several different ways.
   BANANAANANAS *BANANAANANAS* *BANANAANANAS*

   Similarly, the strings *PRODGYRNAMAMMIINCG* and *DYPRONGARMAMMICING* are both shuffles of *DYNAMIC* and *PROGRAMMING*:
   PRODGYRNAMAMMIINCG *DYPRONGARMAMMICING*

   (a) Given three strings $A[1...m]$, $B[1...n]$, and $C[1...m+n]$, describe and analyze an algorithm to determine whether C is a shuffle of A and B.

   (b) **smooth** shuffle of X and Y is a shuffle of X and Y that never uses more than two consecutive symbols of either string. For example,

- *PRDOYGNARAMMMIICNG* is a smooth shuffle of the strings *DYNAMIC* and *PROGRAMMING*.
- *DYPRNOGRAAMMMICING* is a shuffle of *DYNAMIC* and *PROGRAMMING*, but it is not a smooth shuffle (because of the substrings *OGR* and *ING*).
- XXXXXXXXXXXXXXXXXX is a smooth shuffle of the strings *XXXXXXX* and *XXXXXXXXXXX*.
- There is no smooth shuffle of the strings *XXXX* and *XXXXXXXXXXXX*.

Describe and analyze an algorithm to decide, given three strings X, Y, and Z, whether Z is a smooth shuffle of X and Y .

3) Ejercicio 6.8: *The City of Zion* (Kleinberg & Tardos, página 319).

The residents of the underground city of Zion defend themselves through a combination of kung fu, heavy artillery, and efficient algorithms. Recently they have become interested in automated methods that can help fend off attacks by swarms of robots.

Here's what one of these robot attacks looks like.

- A swarm of robots arrives over the course of n seconds; in the i- th second, x_i robots arrive. Based on remote sensing data, you know this sequence $x_1, x_2 ..... x_n$ in advance.
- You have at your disposal an electromagnetic pulse (EMP), which can destroy some of the robots as they arrive; the EMP's power depends on how long it's been allowed to charge up. To make this precise, there is a function $f(\_)$ so that if j seconds have passed since the EMP was last used, then it is capable of destroying up to $f(j)$ robots.
- So specifically, if it is used in the k-th second, and it has been j seconds since it was previously used, then it will destroy $min(x_k, f(j))$ robots. (After this use, it will be completely drained).
- We will also assume that the EMP starts off completely drained, so if it is used for the first time in the j-th second, then it is capable of destroying up to f(j) robots.

The problem. Given the data on robot arrivals $x_1, x_2 ..... x_n$, and given the recharging function $f(\_)$, choose the points in time at which you're going to activate the EMP so as to destroy as many robots as possible.

Example. Suppose $n = 4$, and the values of x_i and $f(i)$ are given by the following table.

i	**1**	**2**	**3**	**4**
x_i	1	10	10	1
$f(i)$	1	2	4	8

The best solution would be to activate the EMP in the 3rd and the 4th seconds. In the 3rd second, the EMP has gotten to charge for 3 seconds, and so it destroys $min(10, 4) = 4$ robots; In the 4th second, the EMP has only gotten to charge for 1 second since its last use, and it destroys $min(1, 1) = 1$ robot. This is a total of **5**.

(a) Show that the following algorithm does not correctly solve this problem, by giving an instance on which it does not return the correct answer.

Schedule-EMP$(x_1, ..., x_n)$

Let j be the smallest number for which $f(j) \geq x_n$
if no such j exists **then**
  set $j = n$
end if
Activate the EMP in the n^{th} second
if $n - j \geq 1$ **then**
  Continue recursively on the input $x_1, \ldots, x_{n-j}$
  (i.e., invoke Schedule-EMP$(x_1, \ldots, x_{n-j})$)
end if

In your example, say what the correct answer is and also what the algorithm above finds.

(b) Give an efficient algorithm that takes the data on robot arrivals $x_1, x_2 ..... x_n$, and the recharging function $f(\_)$, and returns the maximum number of robots that can be destroyed by a sequence of EMP activations.

Solución

1) *Text Segmentation*

2) *Suffle of Strings*

3) *The City of Zion*

(a) Para mostrar el error en este algoritmo, se propone un escenario pequeño, pero que será suficiente para la prueba:

i	**1**	**2**	**3**
x_i	1	4	5
$f(i)$	2	2	1

En este caso, iniciamos verificando si existe un número j tal que $f(j) \geq x_3$ y como ninguno de los valores en $f(\_)$ es mayor o igual que 5, entonces nuestro $j = 3$. Activamos el EMP en el segundo 3 con $f(3)$ y x_3, teniendo un resultado de $min(5,1) = 1$ robot destruido. La condición para poder recurrir es que $n - j \geq 1$, pero vemos que $3 - 3 < 1$, por lo que no podemos recurrir y nuestro resultado final es 1 robot destruido.

Pero, si observamos, esta no sería la respuesta óptima, ya que, si hacemos llamados en los 3 segundos tenemos que: $min(f(1), x_1) + min(f(1), x_2) + min(f(1), x_3) = min(2,1) + min(2,4) + min(2,5) = 1 + 2 + 2 = 5$ robots destruidos, donde se destruyen más que en el resultado dado por el algoritmo.

(b) **Especificación del problema**

Entrada: Un arreglo $X[0, ..., N]$ que representa la cantidad de robots que llegan al campo de batalla por cada segundo y un arreglo $F[0, ..., N]$ que representa cuántos robots puede destruir el EMP con cada segundo de carga.

Salida: La cantidad máxima de robots que se pueden destruir con las activaciones del EMP.

Función objetivo

Sea $0 \leq n \leq N$ y $0 \leq c \leq N$, donde c representa la cantidad de seguntos de carga que tiene el EMP y n la cantidad de segundos totales transcurridos.

$\phi(n, c)$ = Cantidad máxima de robots que se pueden destruir en n segundos con c segundos de carga del EMP.

Reformulación de la función objetivo

Entrada: Un arreglo $X[0,...,N]$ que representa la cantidad de robots que llegan al campo de batalla por cada segundo y un arreglo $F[0,...,N]$ que representa cuántos robots puede destruir el EMP con cada segundo de carga.

Salida: $\phi(0,0)$

Planteamiento Recurrente

$$\phi(n,c) = \begin{cases} 0, & n = N. \\ \uparrow \{\phi(n+1,c+1),\ \ \phi(n+1,0) + min(X[n],F[c])\}, & n < N. \end{cases}$$

Implementación

```
def phi(n, c, X, F, mem):
    ans = None
    if (n,c) in mem: ans = mem[(n, c)]
    else:
        if n == len(X): ans = 0
        else:
            ans = max(phi(n+1, c+1, X, F, mem), phi(n+1, 0, X, F, mem)
   + min(X[n], F[c]))
        mem[(n,c)] = ans
    return ans
```